Problems

1. Task 1

   To get a single embedding for the whole sentence from the transformer's token outputs, we need a pooling strategy. The main options could be

   - `[CLS]`: the final embedding mapping to [CLS] token
   - mean pooling: averaging all the token embeddings
   - max pooling: taking the maximum value across tokens for each feature

   We typically normalize these pooled embeddings so they all have the same scale. This lets us use cosine similarity effectively, focusing just on the direction (angle) between embeddings to gauge sentence similarity, rather than their potentially noisy lengths.

   Considering these methods for sentence similarity:

   Max pooling can be too focused on single strong features and might miss the overall sentence meaning. **Mean pooling** considers all tokens (excluding padding with mask) equally by averaging them, which is a reasonable baseline. **CLS pooling** uses the embedding of the `[CLS]` token, which models like BERT are specifically trained to use as a summary of the entire input sequence, processed through all network layers.

   Given that the `[CLS]` token is designed within the transformer architecture to capture sequence-level information, **CLS pooling is chosen**. This approach leverages the model's learned aggregation mechanism, potentially providing a more nuanced representation than a simple average or a max operation. It directly uses the summary representation the model was trained to produce.

2. **Shared embedding backbone and task-specific heads**

   The multi-task architecture utilizes the sentence embedding model from Task 1 as a shared backbone. This backbone generates a fixed-size sentence embedding from the input, which is then fed into separate layers for each task. This approach enhances parameter and runtime efficiency, as the computationally intensive sentence encoding is done only once.

   L2 normalization on the shared embedding, which was useful in Task 1 for cosine similarity, is omitted here. The task-specific linear layers downstream can adapt to the scale of the input features during training, making prior normalization less critical for these classification heads.

   For Task A (Classification) and Task B (Sentiment Analysis), separate, single linear layers serve as task-specific "heads". Each head takes the shared sentence embedding and projects its dimension to the number of output classes required for that task, producing raw output scores (logits).

During training, these logits can be directly compared against the ground truth labels for each task using a standard cross-entropy loss function to update the model weights.

3. Task 3 Scenario considerations:

   (a) If the entire network should be frozen: Nothing can be learned by freezing the whole network.

   (b) If only the transformer backbone should be frozen: It could be a good option with a pre-trained network we only need to train task-specific heads for each task. Also good for fast iterations in low-data scenarios. Besides, pre-trianing model backbone is retrieved from Terabyte data, using limited data could easily overfit the backbone model.

   (c) If only one of the task-specific heads (either for Task A or Task B) should be frozen: This solution seems not to make sense. Let's say Head A is converged. And then train the backbone and Head B with frezzed Head A. This could lead to the feature drift for Head A. The more reasonable strategy is to freeze both backbone and Head A, only train Head B. Then we can improve Task B's performance without sacrificing Task A's performance

Model choices

Choosing the Pre-trained Model

Selecting the appropriate pre-trained model is a critical first step, balancing potential performance gains from model specificity against training cost and effort. A reasonable sequence for considering model types, from most specific to most general, is as follows:

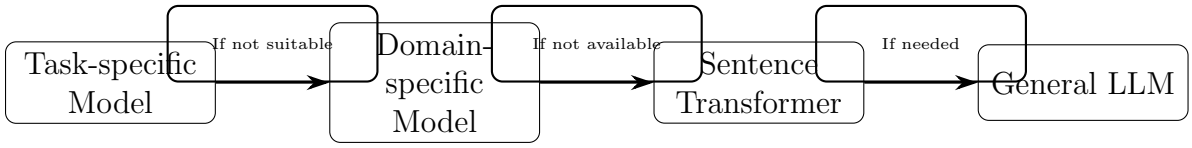
(a) **Task-specific Models:** If a high-quality model already fine-tuned for one of the target tasks (e.g., sentiment classification) exists, using it can be the most direct path. It often provides established benchmarks and relevant learned features.

(b) **Domain-specific Models:** If task-specific models aren't available or suitable for all tasks, models pre-trained on datasets from the relevant domain (e.g., scientific literature, financial news, medical text) are the next best choice. They possess valuable domain-specific vocabulary and contextual understanding.

(c) **Sentence Transformers:** Models specifically fine-tuned for producing general-purpose sentence embeddings (like those from the `sentence-transformers` library) are a strong option. While potentially trained on general domain text, they are optimized for sentence-level understanding, aligning well with the input format for our tasks.

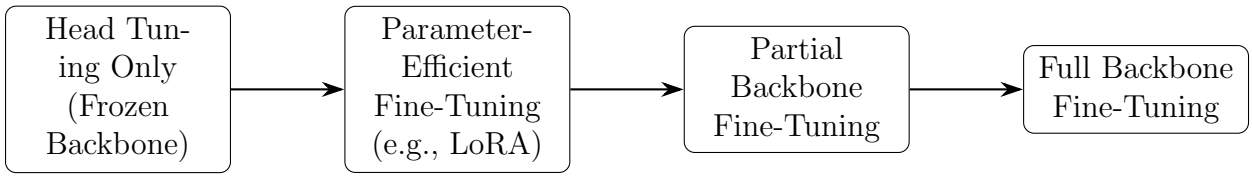
(d) **General Language Models:** Foundational models (like BERT, RoBERTa, etc.) pre-trained on vast, general corpora serve as a fallback. They require the most adaptation but contain broad language knowledge.

This hierarchy can be visualized as:



Fine-tuning Strategies

The choice of base model often informs the required fine-tuning strategy. These strategies vary in the number and type of parameters updated, impacting computational cost, memory requirements, and the degree of adaptation to the target tasks. Generally, the scope of fine-tuning increases as follows:



Key fine-tuning approaches include:

- **Head Tuning Only:** This involves freezing the entire pre-trained backbone and only training the parameters of the newly added task-specific heads (our linear layers). It is the fastest and least memory-intensive method, offering minimal adaptation but preserving the original backbone knowledge. Ideal for small datasets or quick baselines.
- **Parameter-Efficient Fine-Tuning (PEFT), e.g., LoRA:** Techniques like Low-Rank Adaptation (LoRA) involve freezing the backbone but injecting small, trainable "adapter" modules (low-rank matrices) into specific layers (often attention mechanism weights like Q, K, V). Only the parameters of these adapters are trained. This allows adaptation beyond just the heads while keeping the number of trainable parameters much lower than full fine-tuning, offering a good balance between performance and efficiency.
- **Partial Backbone Fine-Tuning:** Here, some layers of the transformer backbone are unfrozen and trained along with the task heads, while the remaining (typically lower) layers stay frozen. Often, the last few encoder layers are chosen for unfreezing. This provides more adaptation capacity than PEFT methods but requires more resources.

- **Full Backbone Fine-Tuning:** In this approach, all parameters of the pre-trained backbone and the task heads are unfrozen and trained. This offers the maximum potential for adaptation to the target data but is the most computationally expensive, requires larger datasets to avoid overfitting, and needs careful hyperparameter tuning (like lower learning rates for the backbone) to prevent catastrophic forgetting.

The optimal strategy depends on the chosen base model's relevance to the tasks, the size and quality of the training data, and available computational resources.

4. Task 3 & 4 insights

   The selection of pre-trained models and the corresponding fine-tuning strategy is highly dependent on the specific application goals and the current development phase.

   Proof-of-Concept (PoC) Phase

   During the PoC phase, the primary objective is typically rapid iteration and validation with minimal initial investment. To achieve speed and efficiency, the preferred strategy often involves combining a readily accessible pre-trained model with the simplest fine-tuning method: **Head Tuning Only**. This means freezing the backbone model and training only the newly added task-specific layers. For the model choice, this might be an easily available **Task-specific model** or a **General pre-trained model** used purely as a feature extractor. This overall approach minimizes development time and computational cost, allowing for quick assessment of feasibility and baseline performance using the model's existing representations.

   Production Phase

   Transitioning to production shifts priorities towards optimizing performance, reliability, cost-effectiveness, and maintainability. Concerns about inference cost, latency, and model controllability may favor well-understood **open-source models** over potentially expensive or opaque large-scale alternatives, offering greater transparency and control. Achieving optimal performance in this phase usually requires more significant adaptation than head tuning alone. Consequently, strategies like **Parameter-Efficient Fine-Tuning (PEFT, e.g., LoRA)**, **Partial Backbone Fine-Tuning**, or **Full Backbone Fine-Tuning** become relevant, with the specific choice depending on the desired performance uplift versus acceptable operational costs.

   The feasibility of more intensive fine-tuning, particularly Partial or Full methods, is strongly linked to the availability of sufficient high-quality training data; without it, aggressive tuning risks overfitting. Furthermore, when fine-tuning the backbone, it is crucial to balance adaptation to the target tasks with the preservation of the model's general language understanding. Overly aggressive fine-tuning on specific tasks can lead to **catastrophic forgetting**, harming generalizability. To mitigate this, common practices include using **differential learning rates**—applying a significantly lower

learning rate to the backbone layers compared to the task-specific heads—and carefully controlling the duration or extent of backbone fine-tuning. This careful approach aims to optimize current task performance while preserving the valuable, general representations learned during the model's pre-training.